Puschmann, André; Kalil, Mohamed A.;
Mitschele-Thiel, Andreas:

A flexible CSMA based MAC protocol for software defined radios

URN: urn:nbn:de:gbv:ilm1-2015210212

Published OpenAccess: January 2015

Original published in:

Frequenz : journal of RF-engineering and telecommunications. - Berlin : De Gruyter (ISSN 2191-6349). - 66 (2012) 9/10, S. 261-268.
DOI: 10.1515/freq-2012-0048
URL: http://dx.doi.org/10.1515/freq-2012-0048
[Visited: 2015-01-14]

DOI 10.1515/freq-2012-0048 — Frequenz 2012; 66(9–10): 261–268

André Puschmann*, Mohamed A. Kalil and Andreas Mitschele-Thiel

A Flexible CSMA based MAC Protocol for Software Defined Radios

Abstract: In this article, we propose a flexible CSMA based MAC protocol which facilitates research and experimentation using software define radios. The modular architecture allows to employ the protocol on platforms with heterogeneous hardware capabilities and provides the freedom to exchange or adapt the spectrum sensing mechanism without modifying the MAC protocol internals. We discuss the architecture of the protocol and provide structural details of its main components. Furthermore, we present throughput measurements that have been obtained on an example system using host-based spectrum sensing.

Keywords: SDR, CSMA, Iris, GNU radio, USRP

PACS® (2010). 84.40.Ua

***Corresponding author: André Puschmann:** Ilmenau University of Technology, P.O. Box 100 565, 98684 Ilmenau, Germany
E-mail: andre.puschmann@tu-ilmenau.de
Mohamed A. Kalil: Ilmenau University of Technology, P.O. Box 100 565, 98684 Ilmenau, Germany
E-mail: mohamed.abdrabou@tu-ilmenau.de
Andreas Mitschele-Thiel: Ilmenau University of Technology, P.O. Box 100 565, 98684 Ilmenau, Germany, E-mail: mitsch@tu-ilmenau.de

1 Introduction

Software Defined Radio (SDR) enables researchers to experiment with communication protocols with a flexibility never experienced before. The flexibility however, comes at a cost in terms of increased delays due to the nature of off-the-shelf radio equipment and host computers. Without a doubt, the increased processing time complicates the realization of *Medium Access Control* (MAC) protocols, especially *Carrier Sense Multiple Access* (CSMA) based protocols, which have strict temporal requirements. This fact has motivated researchers to move time critical functions such as the *Clear Channel Assessment* (CCA) mechanism closer to the RF hardware [8, 13] or even implement the entire MAC protocol in hardware [5]. This approach clearly limits the flexibility and convenience of experimenting with new protocols. Moreover, we believe that the impact of SDR on MAC protocols in general and the CCA mechanism of CSMA based protocols in particular, is still not fully understood.

In this article, we therefore describe a flexible CSMA based MAC protocol for SDR which aims at providing a basis for conducting research in this interesting field. The implementation comprises three blocks: the *Core* block which implements the protocol logic, a *Sensing* block which capsules a specific sensing algorithm and a *Sensing controller* which acts as an interface between core and sensing block. The separation in multiple blocks allows to experiment with different CCA strategies such as host-based sensing or an algorithm implemented on a FPGA without modifying the core component. This work discusses the design of the software architecture as well as its implementation on the SDR framework Iris [10]. This includes the core component, which is based on the well-known IEEE 802.11 standard and a host-based sensing component. Furthermore, we present throughput benchmarks that have been obtained through measurements in a practical networking testbed. The rest of the paper is organized as follows: section 2 discusses related work on MAC protocol implementation. In section 3, we briefly describe the basic access scheme of IEEE 802.11 DCF, outline the impact of SDR on CSMA based protocols and introduce the reader to Iris. In section 4, we present our own implementation of a CSMA protocol. Finally, the performance evaluation of the protocol is presented in section 5. We conclude the paper with a description of future work in section 6.

2 Related work

It is widely known that many SDR platforms experience high communication delays due to the connection

This work has been carried out within the International Graduate School on Mobile Communications (Mobicom), supported by the German Research Foundation (GRK1487) and the Carl Zeiss Foundation.

between the radio hardware such as the USRP and the host computer [12] [11]. Many MAC protocols however, have strict temporal requirements which need to be met by the underlying communication system for successful operation. This fact has motivated researchers to either implement parts or even the entire protocol in hardware. In [8], Nychis et al. propose a *split-functionality* architecture that implements time critical functions such as the CCA and acknowledgement mechanism near the radio hardware to overcome the delay issues mentioned above. Thereby, the functions are controlled from the host CPU through an API. This approach provides a good trade-off between flexibility and achievable performance. However, the realization and any further adaptation of the time critical functions requires a high amount of low-level modifications. In [5], Ansari et al. describe a flexible framework for developing MAC protocols using a hardware-software co-design approach. The authors have analysed different MAC protocols and have identified common functionalities among them, such as a *Timer*, *Carrier Sensing*, *Random Backoff* and *Send Packet*. New protocols may be designed by using a graphical interface to connect these functional blocks together. As all blocks are implemented on the FPGA, any modification to them also requires detailed knowledge in hardware description languages and increases the prototyping time to experiment with new protocol ideas.

3 Basics

3.1 802.11 MAC protocol

This section briefly summarizes the *basic access* mechanism of IEEE 802.11 *Distributed Coordination Function* (DCF) as standardized in [9]. IEEE 802.11 DCF is a contention-based MAC protocol. Unlike reservation-based protocols, which know in advance when to transmit frames, nodes that have data packets to send first have to contend for the channel. This is usually achieved through channel sensing, also known as *Clear Channel Assessment* (CCA), before sending a packet. When the medium is found to be idle for a certain period of time, i.e. *DCF Inter Frame Space* (DIFS), the node starts transmitting the packet. In case the channel is found busy during that time, the node starts a backoff algorithm which defers the transmission for a random amount of time. Before entering the backoff algorithm, the backoff counter is uniformely chosen between zero and the current *contention window* (*CW*). For the first transmission of a packet, *CW* is set to CW_{min}. The size of the window is doubled for every failed transmission until it reaches CW_{max}. During contention, the backoff counter is decremented if the channel is sensed idle for a *slot time*. It is frozen, when the channel is sensed busy but is reactivated when the channel is free again for more than a DIFS amount of time. A network node transmits a packet after its backoff counter has reached zero. After a node has transmitted a packet, it waits for the receiver to acknowledge the packet. If the transmitter does not receive an *ACK* packet within a certain amount of time (i.e. *ACK timeout*), it retransmits the packet again. If a network node successfully receives a packet, it waits for a *Short Inter Frame Space* (SIFS) until it transmits an ACK back to the source of the packet. The backoff algorithm is an important aspect of the protocol which attempts to achieve fairness among network peers and to reduce the probability of packet collision if another node is already transmitting.

From the MAC protocol point of view, the two physical layer characteristics of primary concern are slot time and SIFS. Both values depend on physical and MAC layer components and are provided for each mode of operation in the 802.11 specification.

According to the standard, slot time is defined as:

$$\begin{aligned} aSlotTime = {} & aCCATime + aRxTxTurnaroundTime \\ & + aAirPropagationTime \\ & + aMACProcessingDelay \end{aligned} \tag{3.1}$$

where *aCCATime* is the time required to determine the state of the channel and *aRxTxTurnaroundTime* is the duration to switch from receive to transmit mode. The same document defines SIFS as:

$$\begin{aligned} aSIFSTime = {} & aRxRFDelay + aRxPLCPDelay \\ & + aMACProcessingDelay \\ & + aRxTxTurnaroundTime \end{aligned} \tag{3.2}$$

In other words, slot time includes all physical and MAC layer delays as well as the air propagation time. SIFS is the time the physical and MAC layer of a node requires *"to receive the last symbol of a frame at the air interface, process the frame, and respond with the first symbol on the air interface of the earliest possible response frame"* [9]. The duration of DIFS can be derived from the above mentioned values by the following equation: $DIFS = aSIFSTime + 2 \cdot aSlotTime$. Table 1

Parameter	Value
Slot time	20
SIFS	10
DIFS	50

Table 1: MAC protocol variables defined in IEEE 802.11b (in μs).

summarizes the protocol values as defined in IEEE 802.11b.

3.2 Impact of SDR on MAC parameters

In the IEEE 802.11 standard, all physical and MAC layer parameters are set assuming a certain network scenario, for example a maximum distance between nodes, as well as certain hardware capabilities at each node (i.e. processing power). Low-cost SDR hardware setups suffer from high communication latencies introduced through the communication bus between the RF hardware and the host computer[1].

Figure 1 depicts the packet processing delay in conventional wireless communication system and SDR solutions. In conventional systems such as a 802.11 NIC (see Figure 1(a) the delay between receiving a signal at the air interface and processing the corresponding packet at the MAC layer is typically in the order of microseconds. This is mainly due to highly optimized application specific and integrated hard- and software designs. On the other hand, high bus latencies between digital radio backend and host-based *Digital Signal Processing* (DSP) blocks as well as the low computational power of general purpose processors introduce significant delays in many SDR setups. Figure 1(b) illustrates the impact of SDR on the CCA mechanism employed in CSMA protocols. In this example, the sum of all individual delays exceeds the transmit time of the packet. Hence, the channel status may have already changed by the time the MAC protocol gets the result of a CCA cycle.

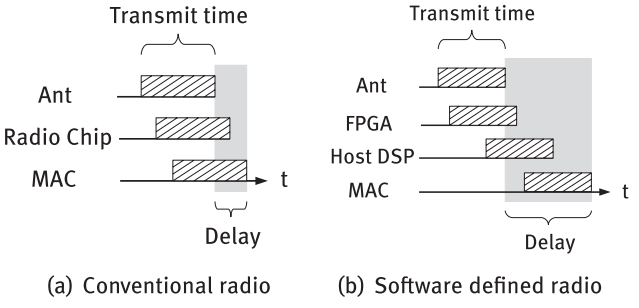

Fig. 1: Antenna to MAC delay in conventional and software defined radios (derived from [12]).

1 For example, the widely used *USRP N or B series* products developed by Ettus Research employ Ethernet or USB connections, respectively, to connect the host PC.

3.3 Iris

Iris [10] is a software framework for designing reconfigurable, component-based SDRs. XML files are used to define the components a radio consists of and how they are connected with each other. Upon start, the Iris core application loads the configuration file and constructs the radio flow graph by connecting input and output ports as specified by the user. The architecture defines multiple building-blocks which may be used to describe an entire radio. The core blocks of the Iris architecture can be briefly described as follows:

- **Component:** Components are self-contained entities implementing a discrete radio function such as a filter, modulator or even an entire MAC protocol. They are characterized by a set of input and/or output ports which are used to connect them with one another.
- **Engine:** Engines are an abstract concept to encapsulate one or more components within a radio. In fact, an engine defines how the specific part of the flow-graph under its regime is executed, how data is passed between components and how the reconfiguration is organized. A radio may consist of one or more engines. Two types of engines are currently defined, namely the *PN engine* and the *Stack engine*. A PN engine is best suited to implement the physical layer of a radio waveform. A single thread of execution sequentially calls all components within the engine according to the data flow defined in the radio configuration. In contrast to that, the Stack engine supports higher layer of the protocol stack of a radio such as the data link or network layer. They are characterized by the fact that they operate on packet granularity rather than on data chunks. Moreover, they are likely to consume and produce packets at the same time and have a bi-directional data flow. In contrast to a PN engine, each component of a Stack engine has its own thread of execution.

– **Controller:** During runtime, controllers are the managing entities of a radio. They are capable to reconfigure all component parameters as well as to change the structure of the entire radio by inserting or deleting components and links. Controller activity is usually triggered by *events* sent out by components.

As described previously, events are usually used to trigger a radio reconfiguration. However, this is not the only use case for events. In response to an event, a controller may also trigger a *command* which can be received by another component. Event and command objects are flexible enough to carry arbitrary user data. Therefore, using the event/command mechanism, light-weight inter-component communication can be realized in Iris. This permits researchers to implement complex radio architectures such as CSMA protocol which requires frequent component interaction. This messaging mechanism is a unique feature of Iris when compared to other SDR frameworks such as GNU Radio.

4 Protocol architecture

In this section, we describe the software architecture of the CSMA based MAC protocol. The key objective of the design was high flexibility that allows to easily extend the protocol as well as interoperability which allows to use the protocol on different SDR platforms. In particular, the architecture should allow to experiment with different CCA strategies. These goals can only be achieved by breaking down the complexity of the entire system into multiple discrete components with defined interface between each other. The proposed protocol architecture therefore comprises three main blocks: the *Core* block which implements the protocol logic (i.e. the protocol state machine), a *Sensing* block which either implements a specific sensing algorithm or capsules an external sensing mechanism such as an FPGA or any other dedicated hardware. The third component of the architecture is the *Sensing controller* which provides a defined interface between the protocol core and the sensing block. The separation into multiple blocks allows to change a single component such as the sensing block without modifying the protocol core.

Figure 2 shows the flow-graph of an example radio that comprises the protocol core, a host-based sensing component and the sensing controller. The core component and the sensing mechanism will be described in the following paragraph.

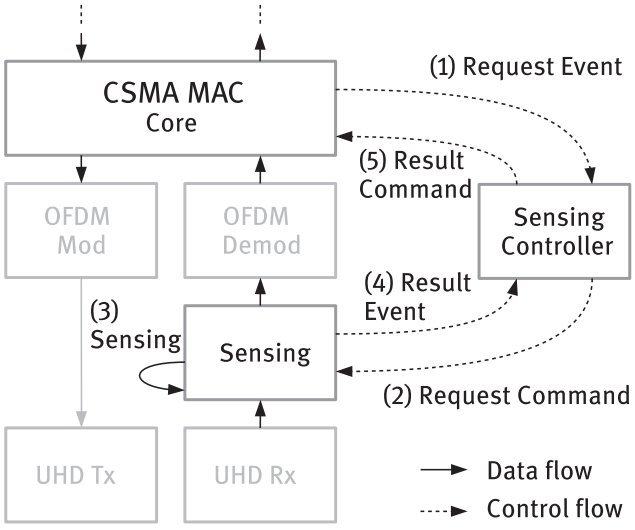

Fig. 2: Waveform of a CSMA based software radio with host-based sensing component.

4.1 CSMA MAC core component

The core component is the heart of the CSMA based MAC protocol. Based on the behaviour of the IEEE 802.11 DCF protocol (described in section 3.1), we have implemented the protocol state machine within an Iris stack component. The component coordinates the packet exchange between two nodes. Through maintaining a sequence number field, the protocol guarantees message order, requests lost frames (retransmission) and eliminates duplicated packets. All protocol values including the maximum number of retransmission and contention window sizes are modifiable through component parameters. The frame format of the CSMA protocol is defined through *Google Protocol Buffers* [2]. From the software developers point of view, this has a number of advantages over the legacy way of defining the frames layout using

```
1  message MacPacket {
2    enum PacketType { DATA = 0; ACK = 1; }
3
4    required string source = 1;
5    required string destination = 2;
6    required PacketType type = 3;
7    optional uint32 seqno = 4;
8    optional bool send_ack = 5;
9    repeated bytes payload = 6;
10 }
```

Listing 1: CSMA frame format definition using Google Protocol Buffers.

byte fields. Protocol buffers can be easily extended, are efficient in terms of space and processing time and simplify data serialization and de-serialization. Listing 1 shows the frame definition of the example system described below (the .proto file).

4.2 Sensing mechanism

To fulfil the flexibility and interoperability goals stated above, we have separated the protocol into three components which communicate with each other over a well-defined interface. For this purpose, we exploit the inter-component communication capabilities of Iris to keep the overhead as low as possible. Figure 2 depicts a complete sensing cycle. If the protocol has a packet to send and reaches the CCA state of the state machine, it issues a sensing *Request Event* to the sensing controller which passes it over to the sensing component as *Request Command*. MAC protocol and sensing component are implemented in different threads of execution. While sensing takes place, the protocol blocks and waits for the *Result Command* to continue operation according to the sensing outcome. After the sensing component has completed the sensing process, it issues a sensing *Result Event* to the sensing controller which includes the outcome of the sensing process (e.g. the computed energy level). Based on this value, the sensing controller determines the channel status by comparing it with a predefined threshold. The channel status is sent back to the MAC protocol as Boolean value.

This design allows to implement the MAC protocol independently from the employed CCA mechanism or decision algorithm. Through a user-definable parameter, the actual sensing component can be easily exchanged, even during runtime.

4.3 System example

To demonstrate the feasibility of the proposed approach, we have implemented a prototype which employs a host-based sensing component. In other words, the example system is fully software based and does not require any additional hardware apart from the USRP. Since sensing is carried out very frequently on a per packet basis, the sensing algorithm employed should be fast to minimize the *Antenna to MAC* delay described in section 3.2. Therefore, the host-based sensing component described in this work uses a simple energy detection algorithm. The sensing component computes the energy level of the received signal in time domain over the whole channel bandwidth based on the incoming samples. The actual sensing time depends on the size of the buffer between UHD Rx and sensing component (see Figure 2) and on the number of iterations. The sensing controller used in this example is threshold based. The channel status can be set by two conditions:

- *Absolute threshold:* The channel is reported as busy if the energy level exceeds a certain threshold and is reported to be free if the energy level is below the fixed threshold.
- *Relative threshold:* The channel is reported as busy if the energy level has increased from one sample to the next and reported to be free if it has decreased by the same value. This mode is useful to detect signals in an environment with time varying noise.

5 Evaluation

We now evaluate the performance of the CSMA protocol described in the previous section. In particular, we study uni- and bidirectional data throughput as well as the fairness of the protocol implementation. First, we describe the experimental setup and how slot time and SIFS have been determined.

5.1 Experimental setup

The setup comprises three SDR nodes, two client nodes plus a third monitor node which is used to determine the protocol parameters. They are located indoors in a laboratory environment separated from each other by 2 m. The center frequency has been set to 2.4 GHz because no other wireless network was active in this band. The host computers used are Core-i5-based systems clocked at 2.53 GHz. Each host is connected to one USRP2. The monitor node runs Matlab for processing the captured data. The client nodes use *Tun/Tap* devices to connect the SDR (i.e. Iris) to the network stack of the operating system. The physical layer of the radio including the OFDM modem has been implemented using *liquid-dsp* [7], a free and open-source digital signal processing library. We refer the reader to [4] for more details about the SDR testbed.

To benchmark the system throughput, we generate constant bit rate network traffic using the network testing tool *iperf* [3]. All measurements are carried out using UDP as transport protocol. Please note that the obtained results

Parameter	Value
RF hardware	USRP2 and XCVR2450
Center frequency	2.401 GHz
Sampling rate	2 Msample/s
Channel bandwidth	2 MHz
Modulation scheme	QPSK
Subcarriers (data)	64 (44)
UHD buffer size	512 B
Transport protocol	UDP
Packet size	100–1500 B
CW_{min}	4
CW_{max}	64
Slot time	3 ms
SIFS	3 ms
DIFS	9 ms

Table 2: Hardware configuration and protocol parameters.

do not consider the overhead introduced at the MAC protocol such as retransmissions which is referred to as *goodput*. The detection threshold of the sensing mechanism has been determined experimentally. The hardware configuration as well as the parameter settings of the protocol are summarized in Table 2.

5.2 Protocol parameter determination

Slot time and *SIFS* are the two main 802.11 DCF protocol parameter. The slot time includes all physical and MAC layer processing delays including CCA and air propagation time. Hence, slot time may be approximated by measuring the delay between the instant at which the channel state turns from busy to idle and the time at which the SDR node actually accesses the channel. For simulating a busy channel, we are using one client SDR node (denoted as *Blocker*) which transmits a continuous pseudo-random OFDM signal. The blocker activity (i.e. *on/off*) can be controlled by the user. The device under test (i.e. the second client node) is configured such that it sends a single packet with a predefined size of 1000 B after the blocker is turned off. The monitoring node captures the channel activity during the experiment. The recorded samples are processed offline through a Matlab script which plots the received energy level (calculated in time domain) vs. time. The slot time duration can be read from the signal plot.

To determine the SIFS, we have used both client nodes. We line break after “signal plot” have configured them such that they were exchanging *DATA* and *ACK* packets. Using the monitor node, we have again captured the radio scene and measured the time between the last symbol of the *DATA* packet and the first symbol of the *ACK* packet.

We have repeated both experiments ten times. The averaged results are shown in Table 3. The table also shows the scaling factor (based on the rounded average) compared to the values defined in the 802.11b standard (see Table 1). In our SDR setup, we observed an average time that was between 200 and 300 times higher compared to a IEEE 802.11b based radio. It is worth pointing out that this delay also involves the packet forming delay which is done after the channel has been reported to be idle. The obtained results are in the region of those reported in [8] for fully host-based SDR setups.

5.3 Unidirectional throughput

In this paragraph, we measure the impact of the packet size on the maximum throughput. In each transmit cycle, we assume that one and only one transmitter is active which always has a non-empty transmit queue. All other stations only receive packets and reply with ACK packets (unidirectional traffic). In each run, we transferred 10 MB and linearly increased the packet size up to 1500 B. The results are depicted in Figure 3. As expected, the observed throughput increases as the packet size increases until it asymptotically approaches its maximum value of 646 kbit/s. This behaviour is explained by considering that a larger packet size only decreases the time spent for transmitting the data frame itself. A considerable amount of time is constant communication overhead spent on acknowledgements or caused by delays introduced by the SDR system.

Parameter	Min	Avg	Max	Sd	Factor
Slot time	3.2	3.4	3.6	0.15	×200
SIFS	2.3	2.5	2.7	0.2	×300

Table 3: Results of slot time and SIFS parameter measurements (in ms).

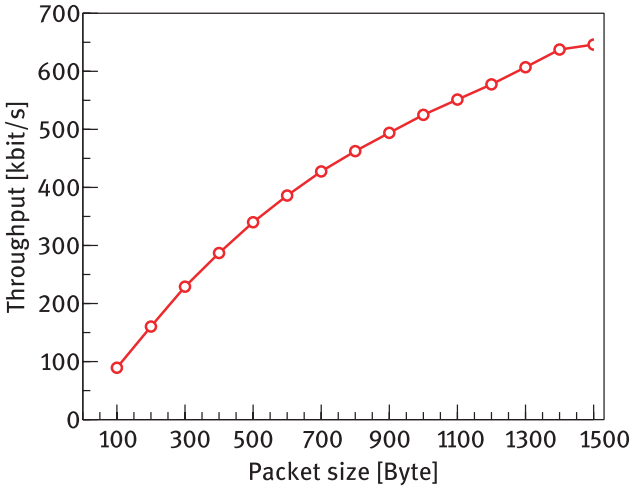

Fig. 3: Unidirectional throughput as a function of packet size.

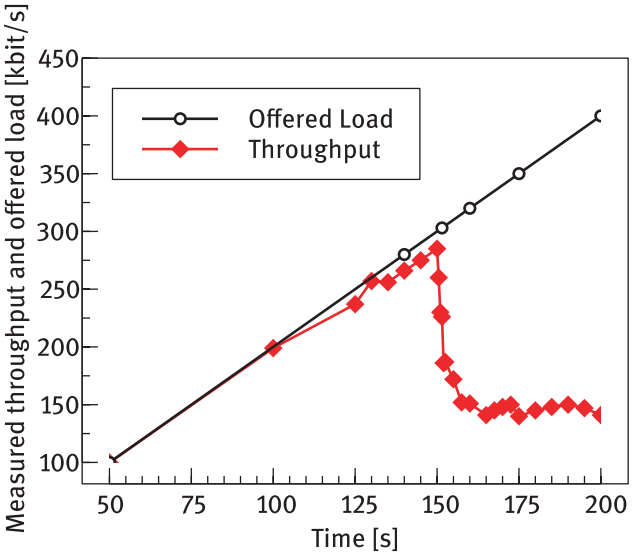

Fig. 4: Throughput with slowly increasing offered load.

5.4 Bidirectional saturation throughput

In this paragraph, we study the *saturation throughput*. This is a fundamental performance figure for contention-based communication systems and is defined as "*the limit reached by the system throughput as the offered load increases, and represents the maximum load that the system can carry in stable conditions*" [6]. In other words, if the offered load (the load produced at application layer) is increased linearly and reaches a certain limit, the system will achieve its *maximum throughput*. Any further increase results in a significant performance degradation until it asymptotically approaches the saturation throughput, the maximum throughput the system can carry in overload conditions.

To determine both, maximum and saturation throughput of the system we ran experiments where the offered load linearly increases with time. The measurements were carried out with the same setup described above (see Table 2). *Iperf* was used to generate bidirectional network traffic (-d parameter).

The results are depicted in Figure 4. The straight line shows the linearly increasing offered load. We can observe that the achieved throughput closely follows the offered load for the first 150 s of the experiment. After reaching its maximum value, the throughput dramatically drops until it asymptotically approaches its saturation value at roughly 150 kbit/s. The system reaches its maximum throughput at 285 kbit/s.

5.5 Fairness

In this section, we evaluate the fairness of the system by counting the number of packets transmitted by each node.

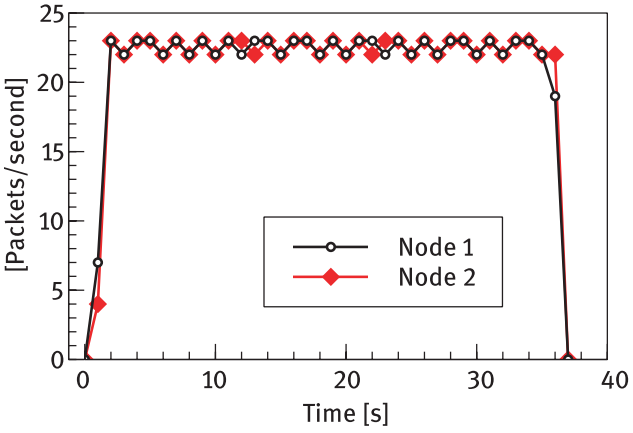

Fig. 5: Fairness measured as successfully transmitted packets per node.

Iperf is again used to generate bidirectional network traffic at a rate of 280 kbit/s (below maximum throughput) using a packet size of 1500 B. The measured packet rate is depicted in Figure 5. The results show that the average number of packets transmitted by each node is almost the same during the whole experiment (36 s). This proves the fairness of the protocol in a non-overloaded situation.

6 Conclusion

In this article, we have described a flexible CSMA based MAC protocol which has been implemented on a reconfigurable SDR testbed called Iris. Our work was motivated by the fact that available CSMA protocols either implement

parts or the entire protocols on the radio hardware. Any modification or adaptation therefore requires a high degree of low-level knowledge and therefore limits the flexibility and convenience of experimenting with new protocols. To overcome this issue, we have designed a protocol architecture that comprises three blocks, the core block which implements the protocol logic, a sensing block which capsules the CCA mechanism and a sensing controller which acts as an interface between core and sensing block. The separation allows to experiment with different sensing mechanisms without modifying the core component. In this article, we have evaluated the protocol performance in terms of data throughput in a practical SDR networking testbed using a host-based sensing component. In the future, we plan to further investigate the impact of different spectrum sensing strategies on the performance of CSMA based MAC protocols. We are currently studying the impact of an external spectrum sensing hardware as part of an experiment within the FP7-CREW project [1].

Received: May 31, 2012.

References

[1] *European Commission Seventh Framework Programme (FP7) Cognitive Radio Experimentation World (CREW)*, Webpage: http://www.crew-project.eu/.

[2] *Google Protocol Buffers*, Webpage: http://code.google.com/p/protobuf/.

[3] *Iperf*, Webpage: http://sourceforge.net/projects/iperf/.

[4] A. Puschmann, M. A. Kalil and A. Mitschele-Thiel, Implementation and Evaluation of a Practical SDR Testbed, in: *4th International Conference on Cognitive Radio and Advanced Spectrum Management*, Barcelona, Spain, October 2011.

[5] J. Ansari, Xi Zhang, A. Achtzehn, M. Petrova and P. Mähönen, A Flexible MAC Development Framework for Cognitive Radio Systems, in: *IEEE Wireless Communications and Networking Conference (WCNC)*, March 2011.

[6] G. Bianchi, Performance Analysis of the IEEE 802.11 Distributed Coordination Function, in: *IEEE Journal on Selected Areas in Communications*, 18, March 2000.

[7] Joseph D. Gaeddert, *Facilitating Wireless Communications through Intelligent Resource Management on Software-Defined Radios in Dynamic Spectrum Environments*, Ph.D. thesis, Virginia Polytechnic Institute & State University, Blacksburg, VA, January 2011.

[8] George Nychis, Thibaud Hottelier, Zhuochen Yang, Srinivasan Seshan and Peter Steenkiste, Enabling MAC Protocol Implementations on Software-defined Radios, in: *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*, 2009.

[9] IEEE Computer Society, IEEE 802.11 Standard, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification, (2007).

[10] P. D. Sutton, J. Lotze, H. Lahlou, S. A. Fahmy, K. E. Nolan, B. Ozgul, T. W. Rondeau, J. Noguera and L. E. Doyle, Iris: An Architecture for Cognitive Radio Networking Testbeds, in: *IEEE Communications Magazine*, 48, September 2010.

[11] Stefan Valentin, Holger von Malm and Holger Karl, *Evaluating the GNU Software Radio platform for wireless testbeds*, Report, 2006.

[12] Thomas Schmid, Oussama Sekkat and Mani B. Srivastava, An Experimental Study of Network Performance Impact of Increased Latency in SDR, in: *ACM Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization (WINTECH)*, 2007.

[13] W. Hu, H. Yousefizadeh and X. Li, Load Adaptive MAC: A Hybrid MAC Protocol for MIMO SDR MANETs, in: *IEEE Transactions on Wireless Communications*, 10, November 2011.